Controllo della ALU

- La ALU è usata per:
  - Load/Store: F = somma
  - Branch: F = sottrazione
  - Tipo R: F dipende dal campo `funct`

Controllo ALU	Funzione
0000	AND
0001	OR
0010	somma
0110	sottrazione
0111	set less than
1100	NOR

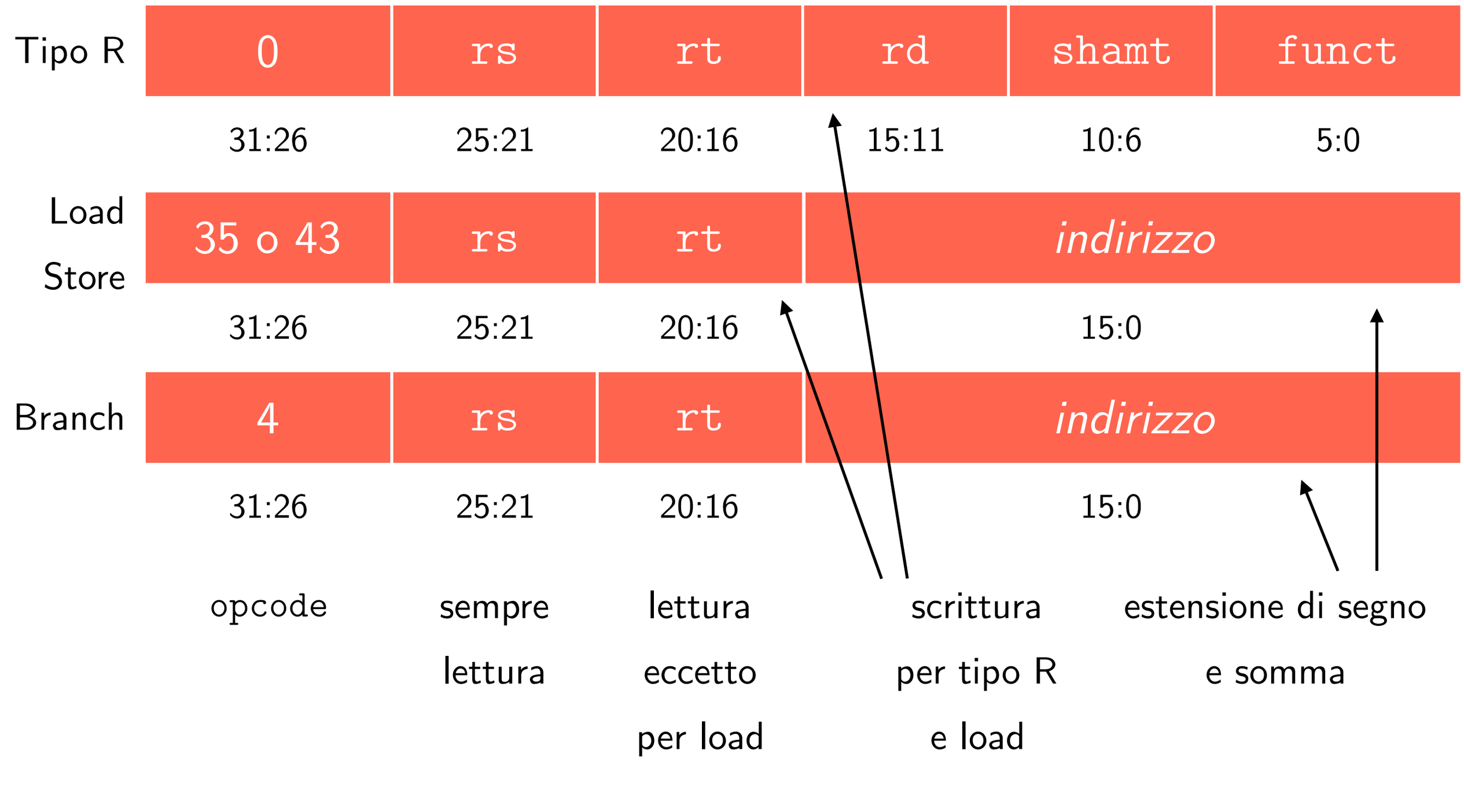
Controllo della ALU

- Si assuma un campo di controllo su 2 bit chiamato ALUOp derivato da opcode
  - Logica combinatoria per calcolare il controllo della ALU

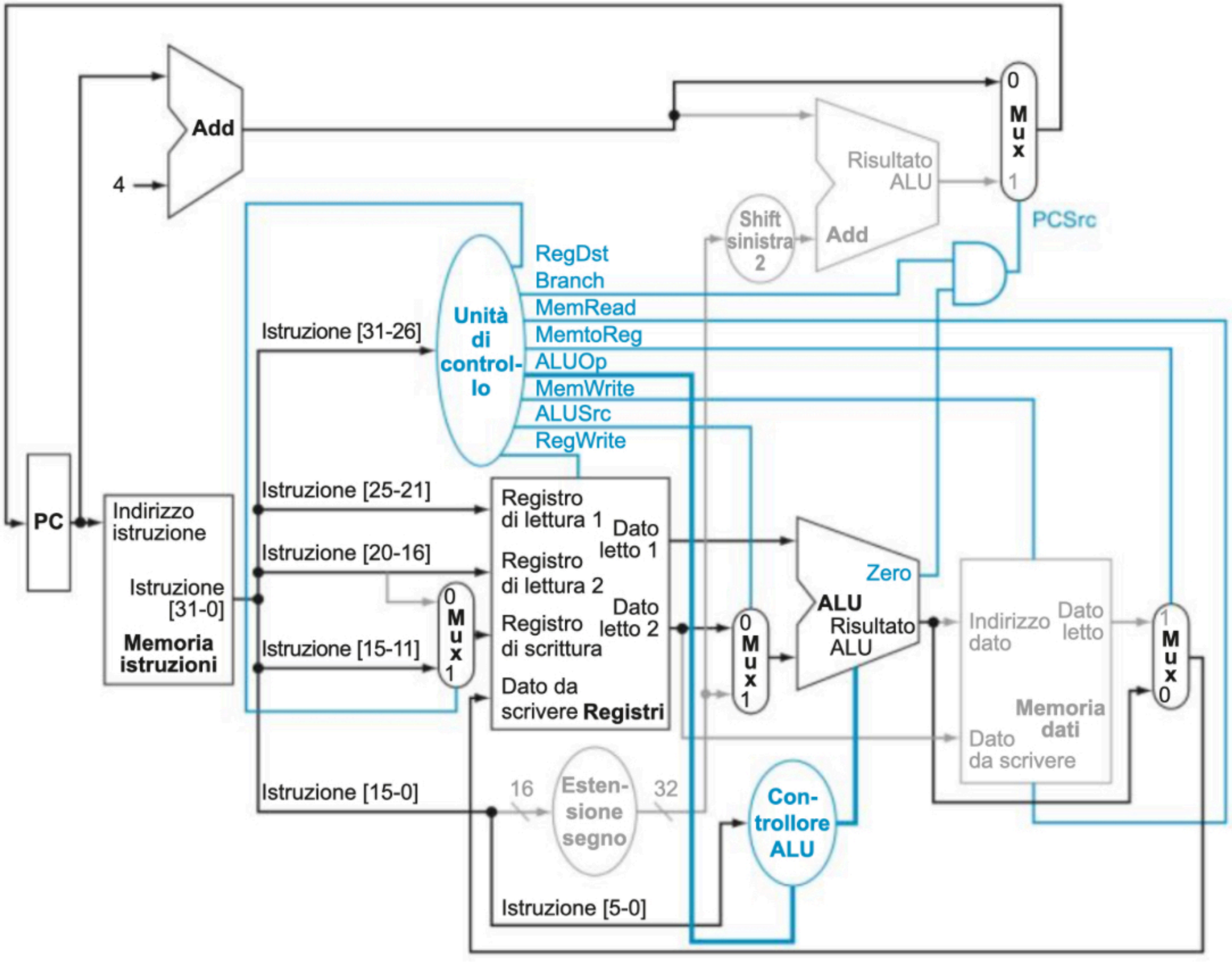
opcode	ALUOp	Operazione	funct	Funzione ALU	ALU control
lw	00	Load word	XXXXXX	somma	0010
sw	00	Store word	XXXXXX	somma	0010
beq	01	Branch equal	XXXXXX	sottrazione	0110
R-type 0111 1100	10	add	100000	somma	0010
		subtract	100010	sottrazione	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

Unità di controllo principale

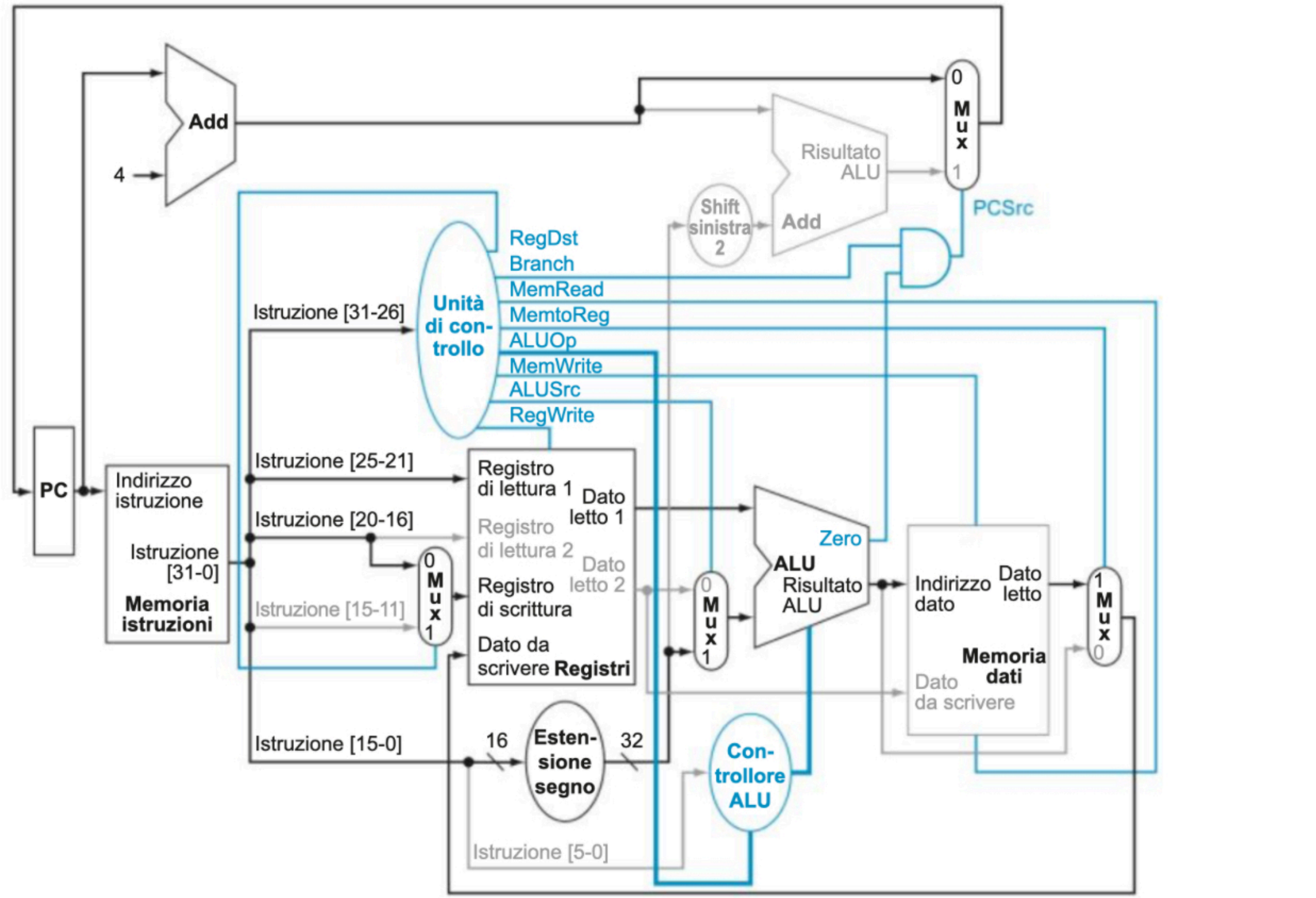
I segnali di controllo sono derivati dall'istruzione



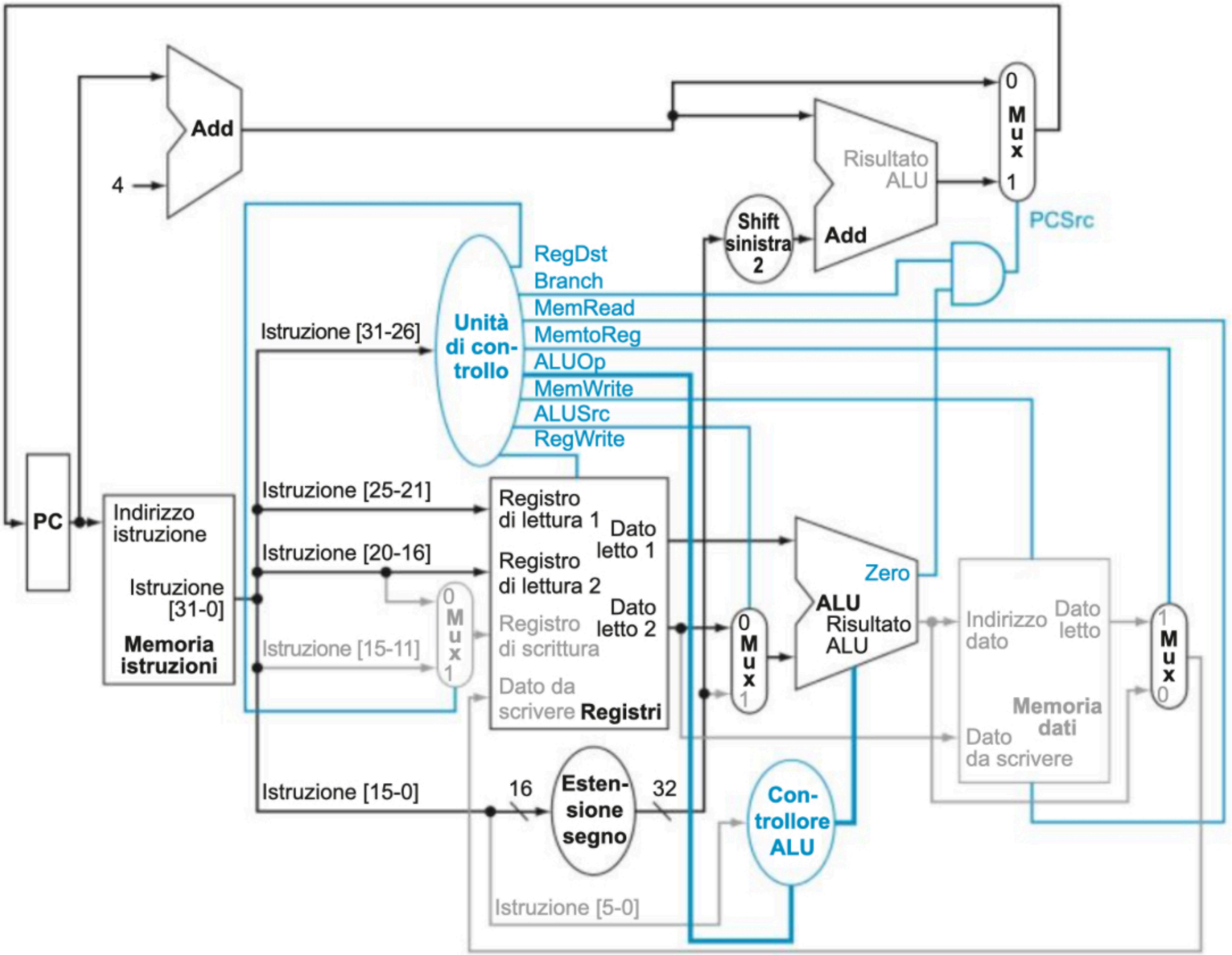
Istruzione di tipo R



Istruzione Load



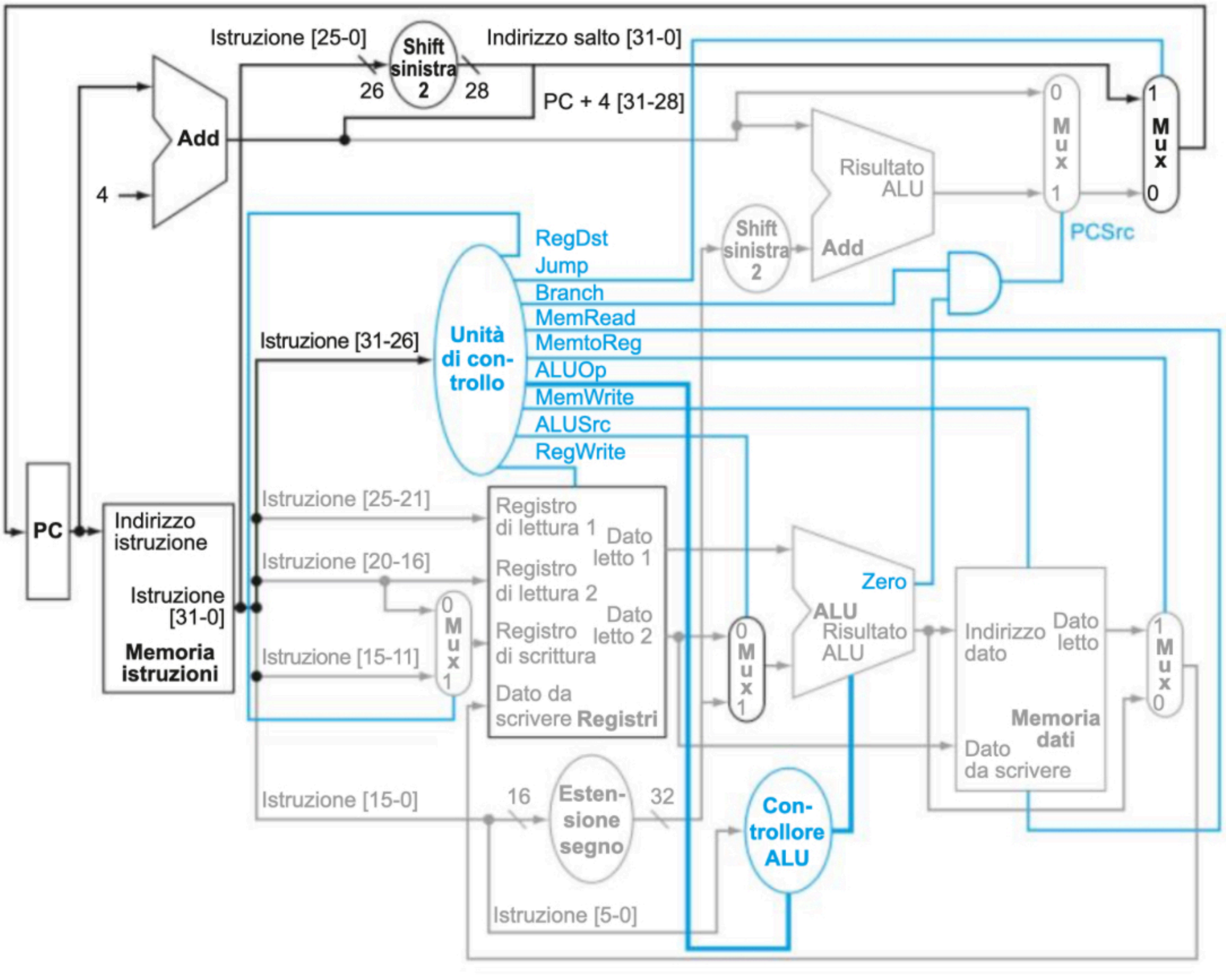
Istruzione Branch-on-Equal



Implementare i salti

- L'istruzione `jump` usa indirizzi di parola
- Aggiornare PC con la concatenazione di
  - I 4 bit più significativi del vecchio valore di PC
  - L'indirizzo di salto su 26 bit
  - `00`
- Necessità di un segnale di controllo extra decodificato da `opcode`

Datapath con aggiunta dei salti

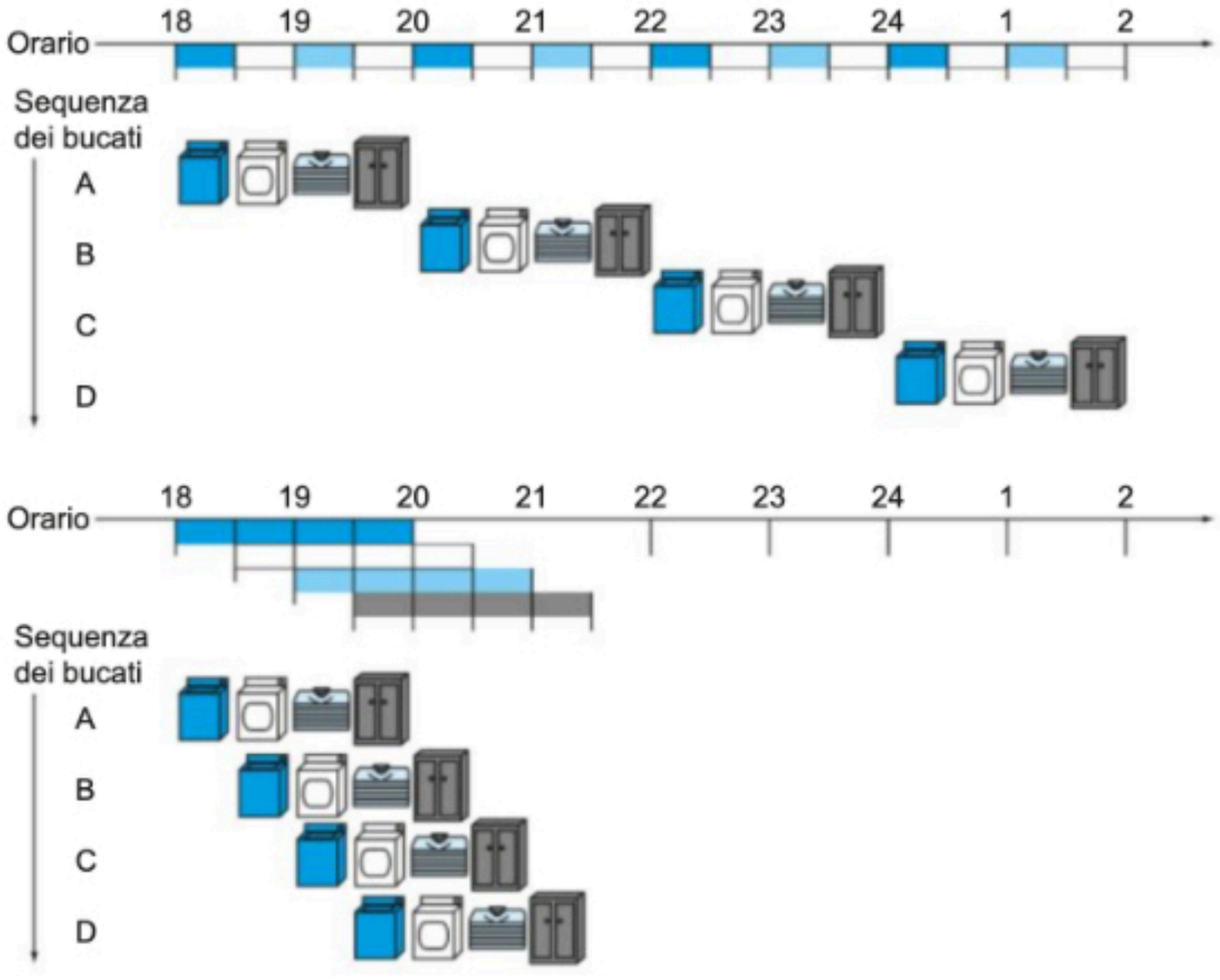


Problemi di prestazioni

- Il ritardo più lungo determina il periodo di clock
  - Percorso critico: istruzione Load
  - Memoria istruzioni $\rightarrow$ Registri $\rightarrow$ ALU $\rightarrow$ Memoria dati $\rightarrow$ Registri
- Non è ammissibile cambiare periodo di clock per istruzioni diverse
- Viola un principio di progetto
  - Rendere il caso comune veloce
- Miglioreremo le prestazioni tramite *pipelining*

Analogia della pipeline

- Lavanderia: sovrapporre esecuzioni
  - Il parallelismo migliora le prestazioni



- Quattro carichi:
  - Speedup
    $= 8 / 3.5 = 2.3$
- Non-stop:
  - Speedup
    $= 2n/0.5n + 1.5 \cong 4$
    $=$ numero di stadi

Pipeline del MIPS

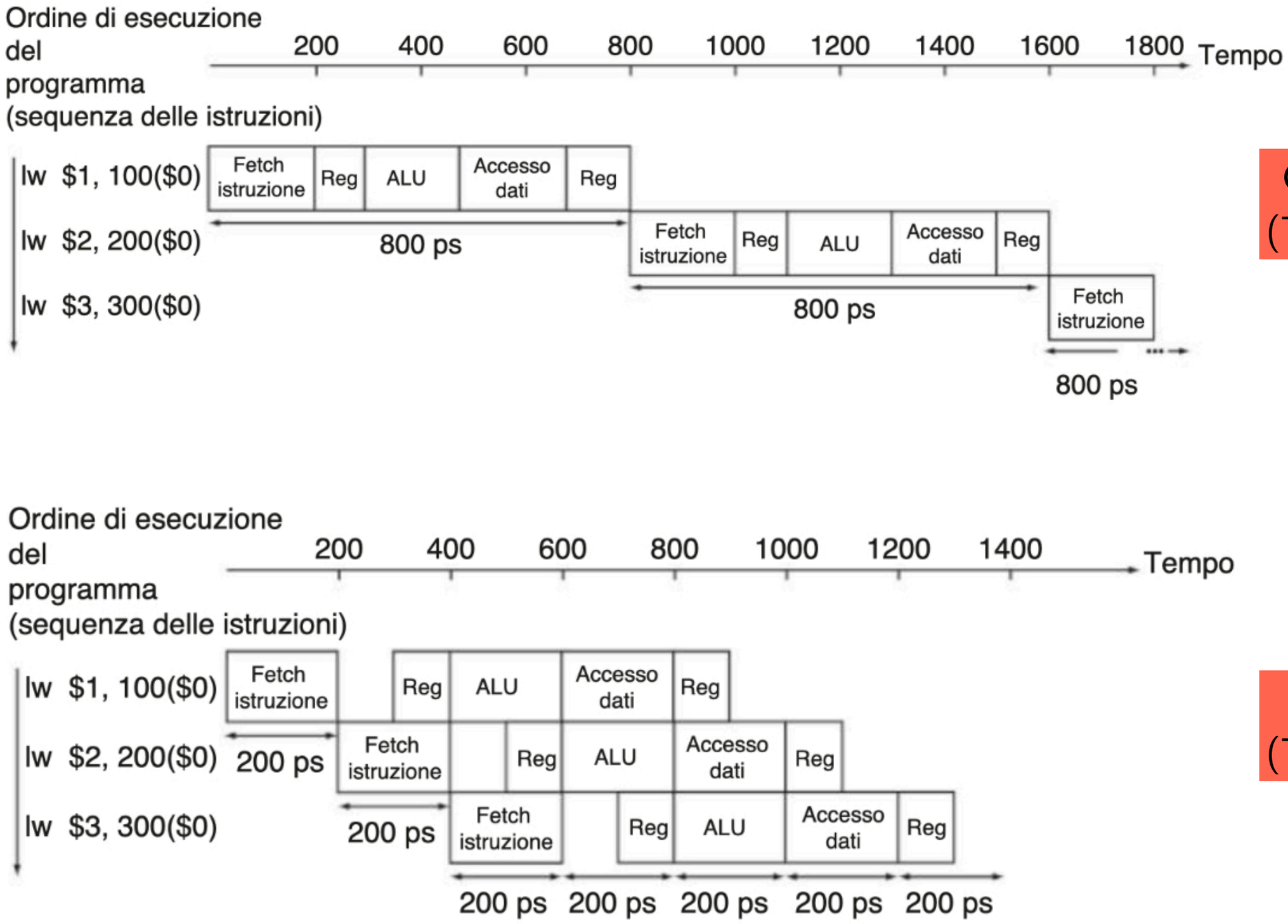
- Cinque stadi, un passo per stadio
  - IF: fetch dell'istruzione dalla memoria istruzioni
  - ID: decodifica dell'istruzione e lettura dei registri
  - EX: esecuzione dell'operazione o calcolo dell'indirizzo
  - MEM: accesso all'operando nella memoria dati
  - WB: scrittura del risultato in un registro

Prestazioni della pipeline

- Si assuma che il tempo di ogni stadio sia:
  - 100 ps per leggere o scrivere un registro
  - 200 ps per tutti gli altri stadi
- Si confrontino le unità di elaborazione in pipeline e a ciclo singolo

Tipo di istruzione	Fetch istruzione	Lettura registri	Operazione ALU	Accesso ai dati in memoria	Scrittura Registri	Tempo Totale
`lw`	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
`sw`	200 ps	100 ps	200 ps	200 ps		700 ps
Tipo R	200 ps	100 ps	200 ps		100 ps	600 ps
`beq`	200 ps	100 ps	200 ps			500 ps

Prestazioni della pipeline



Ciclo singolo
(T_C = 800 ps)

Pipeline
(T_C = 200 ps)

Speedup della pipeline

- Se tutti gli stadi sono bilanciati
  - cioè, tutti richiedono lo stesso tempo
  - Tempo tra le istruzioni$_{\text{CON PIPELINE}}$
    = Tempo tra le istruzioni$_{\text{SENZA PIPELINE}}$/Numero di stadi
- Se sbilanciato, lo speedup è inferiore
  - Latenza (tempo per ogni istruzione) non decresce

Pipeline e progetto dell'ISA

- L'ISA del MIPS è progettata per il pipelining
  - Tutte le istruzioni sono a 32 bit
    - Più facile prelevare e decodificare in un ciclo
    - c.f. x86: istruzioni da 1 a 17 bit
- Pochi e regolari formati di istruzioni
  - Si possono decodificare e leggere i registri in un passo
- Indirizzamento load/store
  - Si può calcolare l'indirizzo nel 3° stadio e accedere alla memoria nel 4° stadio
- Allineamento in memoria degli operandi
  - Un accesso alla memoria richiede un solo ciclo